CENTRAL TEST

March 17, 2024

[21]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import style
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LinearRegression
```

[22]:

```
# Load the dataset
data = pd.read_csv("C:\\Users\\HP\\Desktop\\kiva_mpi_region_locations.csv")
data
```

[22]:

```
                 LocationName  ISO      country      region world_region  \
0     Badakhshan, Afghanistan  AFG  Afghanistan  Badakhshan   South Asia
1        Badghis, Afghanistan  AFG  Afghanistan     Badghis   South Asia
2        Baghlan, Afghanistan  AFG  Afghanistan     Baghlan   South Asia
3          Balkh, Afghanistan  AFG  Afghanistan       Balkh   South Asia
4         Bamyan, Afghanistan  AFG  Afghanistan      Bamyan   South Asia
…                         …   …          …          …           …
2767                      NaN  NaN          NaN         NaN          NaN
2768                      NaN  NaN          NaN         NaN          NaN
2769                      NaN  NaN          NaN         NaN          NaN
2770                      NaN  NaN          NaN         NaN          NaN
2771                      NaN  NaN          NaN         NaN          NaN

        MPI                            geo        lat        lon
0     0.387  (36.7347725, 70.81199529999999)  36.734772  70.811995
1     0.466         (35.1671339, 63.7695384)  35.167134  63.769538
2     0.300         (35.8042947, 69.2877535)  35.804295  69.287754
3     0.301         (36.7550603, 66.8975372)  36.755060  66.897537
4     0.325         (34.8100067, 67.8212104)  34.810007  67.821210
…       …                              …        …         …
2767    NaN                 (1000.0, 1000.0)        NaN        NaN
2768    NaN                 (1000.0, 1000.0)        NaN        NaN
2769    NaN                 (1000.0, 1000.0)        NaN        NaN
2770    NaN                 (1000.0, 1000.0)        NaN        NaN
2771    NaN                 (1000.0, 1000.0)        NaN        NaN

[2772 rows x 9 columns]
```

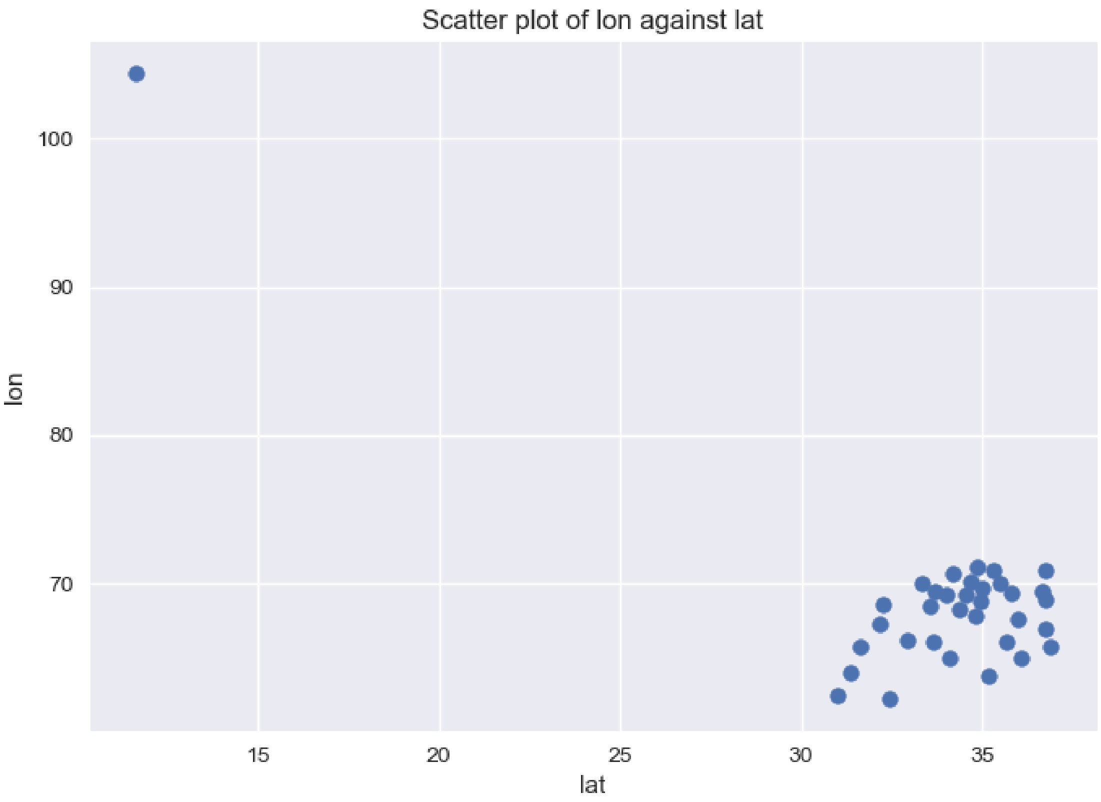
[23]:

```
# Prepare the features and target variable
X = data[['lat']].iloc[:34]  # Considering only the first 34 rows for simplicity
y = data['lon'].iloc[:34]  # Assuming 'lon' as the target variable
```

[24]:

```
# Plot a scatter graph
style.use('seaborn')
plt.xlabel("lat")
plt.ylabel("lon")
plt.scatter(X, y)
plt.title("Scatter plot of lon against lat")
plt.show()
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_9104\488471832.py:2:
MatplotlibDeprecationWarning: The seaborn styles shipped by Matplotlib are
deprecated since 3.6, as they no longer correspond to the styles shipped by
seaborn. However, they will remain available as 'seaborn-v0_8-<style>'.
Alternatively, directly use the seaborn API instead.
  style.use('seaborn')
```



[25]:

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,␣
 ↪random_state=42)
```

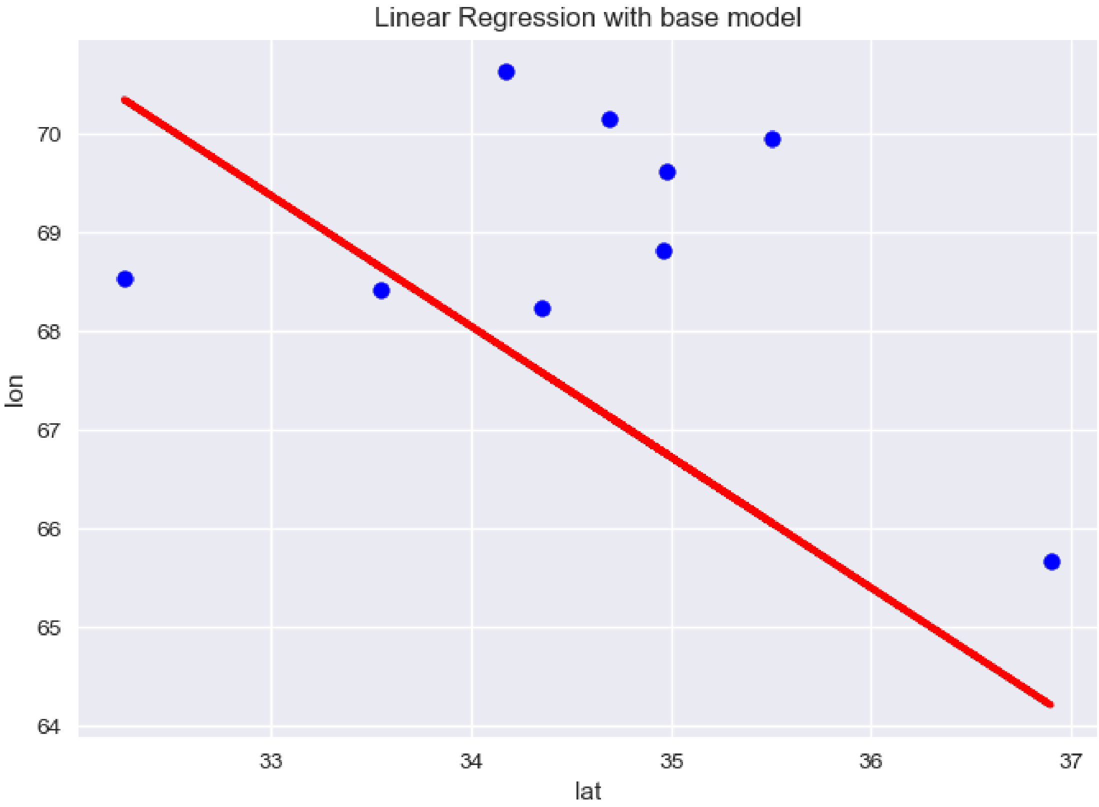
[26]:
```
# Create a linear regression model
base_model = LinearRegression()
```

[27]:
```
# Fit the base model
base_model.fit(X_train, y_train)
```

[27]:
```
LinearRegression()
```

[28]:
```
# Make predictions with the base model
base_pred = base_model.predict(X_test)
```

[29]:
```
# Plot the regression line
plt.scatter(X_test, y_test, color='blue')
plt.plot(X_test, base_pred, color='red', linewidth=3)
plt.xlabel("lat")
plt.ylabel("lon")
plt.title("Linear Regression with base model")
plt.show()
```



[30]:
```
# Calculate the coefficient and intercept of the base model
base_coef = base_model.coef_
```

```
base_intercept = base_model.intercept_
print("Coefficient:", base_coef)
print("Intercept:", base_intercept)
```

```
Coefficient: [-1.32226706]
Intercept: 113.00294245097427
```

[31]:

```
# Calculate the R^2 score of the base model
base_r2_score = base_model.score(X_test, y_test)
print("R^2 Score of base model:", base_r2_score)
```

```
R^2 Score of base model: -1.8921327959431977
```

[32]:

```
# Define parameter grid for grid search
param_grid = {
    'fit_intercept': [True, False],
    'copy_X': [True, False],
    'positive': [True, False]
}
```

[33]:

```
# Perform grid search
grid_search = GridSearchCV(base_model, param_grid, cv=5)
grid_search.fit(X_train, y_train)
```

[33]:

```
GridSearchCV(cv=5, estimator=LinearRegression(),
             param_grid={'copy_X': [True, False],
                         'fit_intercept': [True, False],
                         'positive': [True, False]})
```

[34]:

```
# Get the best parameters found by grid search
best_params = grid_search.best_params_
print("Best Parameters:", best_params)
```

```
Best Parameters: {'copy_X': True, 'fit_intercept': True, 'positive': True}
```

[35]:

```
# Create a new model with the best parameters
optimized_model = LinearRegression(**best_params)
```

[36]:

```
# Fit the optimized model
optimized_model.fit(X_train, y_train)
```

[36]:

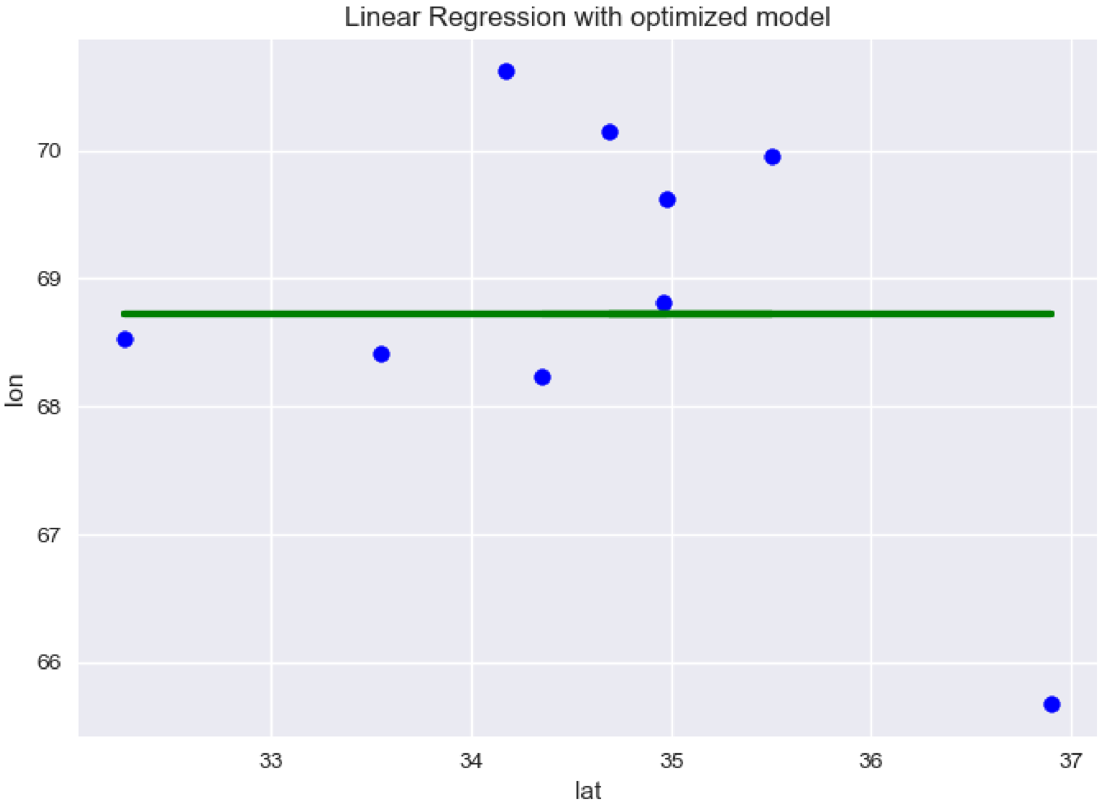
```
LinearRegression(positive=True)
```

[37]:

```
# Make predictions with the optimized model
optimized_pred = optimized_model.predict(X_test)
```

[38]:

```
# Plot the regression line for optimized model
plt.scatter(X_test, y_test, color='blue')
plt.plot(X_test, optimized_pred, color='green', linewidth=3)
plt.xlabel("lat")
plt.ylabel("lon")
plt.title("Linear Regression with optimized model")
plt.show()
```



[39]:
```
# Calculate the coefficient and intercept of the optimized model
optimized_coef = optimized_model.coef_
optimized_intercept = optimized_model.intercept_
print("Optimized Coefficient:", optimized_coef)
print("Optimized Intercept:", optimized_intercept)
```

```
Optimized Coefficient: [0.]
Optimized Intercept: 68.72545600000001
```

[40]:
```
# Calculate the R^2 score of the optimized model
optimized_r2_score = optimized_model.score(X_test, y_test)
print("R^2 Score of optimized model:", optimized_r2_score)
```

```
R^2 Score of optimized model: -0.013815565615129088
```

LOGISTIC REGRESSION

```
[36]: import pandas as pd
      from sklearn.model_selection import train_test_split
      from sklearn.linear_model import LogisticRegression
      from sklearn.preprocessing import KBinsDiscretizer
      from sklearn.metrics import accuracy_score
```

```
[37]: # Load the dataset
      data = pd.read_csv("C:\\Users\\HP\\Desktop\\activitycalorieintensitystepsJoined␣
       ↪- Extract 1.csv")
      data
```

```
[37]:              Id ActivityDate  TotalSteps  TotalDistance  TrackerDistance  \
      0    1624580081     05-01-16       36019      28.030001        28.030001
      1    1644430081    4/14/2016       11037       8.020000         8.020000
      2    1644430081    4/19/2016       11256       8.180000         8.180000
      3    1644430081    4/28/2016        9405       6.840000         6.840000
      4    1644430081    4/30/2016       18213      13.240000        13.240000
      ..          …          …         …            …               …
      935  1844505072    4/20/2016           8       0.010000         0.010000
      936  4020332650    4/17/2016          16       0.010000         0.010000
      937  4319703577     05-12-16          17       0.010000         0.010000
      938  6775888955     05-03-16           9       0.010000         0.010000
      939  7086361926    4/16/2016          31       0.010000         0.010000

           LoggedActivitiesDistance  VeryActiveDistance  ModeratelyActiveDistance  \
      0                         0.0               21.92                      4.19
      1                         0.0                0.36                      2.56
      2                         0.0                0.36                      2.53
      3                         0.0                0.20                      2.32
      4                         0.0                0.63                      3.14
      ..                        …                 …                       …
      935                       0.0                0.00                      0.00
      936                       0.0                0.00                      0.00
      937                       0.0                0.00                      0.00
      938                       0.0                0.00                      0.00
      939                       0.0                0.00                      0.00

           LightActiveDistance  SedentaryActiveDistance  …  \
      0                   1.91                     0.02  …
      1                   5.10                     0.00  …
      2                   5.30                     0.00  …
      3                   4.31                     0.00  …
      4                   9.46                     0.00  …
      ..                   …                       …  …
      935                 0.01                     0.00  …
      936                 0.01                     0.00  …
      937                 0.01                     0.00  …
```

```
938                 0.01                     0.00  …
939                 0.01                     0.00  …

     LightlyActiveMinutes_1  FairlyActiveMinutes_1  VeryActiveMinutes_1  \
0                       171                     63                  186
1                       252                     58                    5
2                       278                     58                    5
3                       227                     53                    3
4                       402                     71                    9
..                        …                      …                    …
935                       1                      0                    0
936                       2                      0                    0
937                       2                      0                    0
938                       1                      0                    0
939                       3                      0                    0

     SedentaryActiveDistance_1  LightActiveDistance_1  \
0                         0.02                   1.91
1                         0.00                   5.10
2                         0.00                   5.30
3                         0.00                   4.31
4                         0.00                   9.46
..                           …                      …
935                       0.00                   0.01
936                       0.00                   0.01
937                       0.00                   0.01
938                       0.00                   0.01
939                       0.00                   0.01

     ModeratelyActiveDistance_1 VeryActiveDistance_1      Id_1_1  \
0                          4.19                21.92  1624580081
1                          2.56                 0.36  1644430081
2                          2.53                 0.36  1644430081
3                          2.32                 0.20  1644430081
4                          3.14                 0.63  1644430081
..                            …                    …           …
935                        0.00                 0.00  1844505072
936                        0.00                 0.00  4020332650
937                        0.00                 0.00  4319703577
938                        0.00                 0.00  6775888955
939                        0.00                 0.00  7086361926

     ActivityDay_1 StepTotal
0         05-01-16     36019
1        4/14/2016     11037
2        4/19/2016     11256
3        4/28/2016      9405
```

```
4        4/30/2016     18213
..             …         …
935      4/20/2016         8
936      4/17/2016        16
937       05-12-16        17
938       05-03-16         9
939      4/16/2016        31

[940 rows x 31 columns]
```

[38]:

```python
# Drop any non-numeric columns or columns causing errors (like date columns)
# For simplicity, we'll drop all non-numeric columns for now
numeric_data = data.select_dtypes(include='number')
```

[39]:

```python
# Split the data into features (X) and target variable (y)
X = numeric_data.drop(columns=['TotalDistance'])  # Features
y = numeric_data['TotalDistance']  # Target variable
```

[40]:

```python
# Convert the target variable into categories or bins
# Here, we'll use KBinsDiscretizer to convert it into 5 bins
est = KBinsDiscretizer(n_bins=5, encode='ordinal', strategy='uniform')
y_bins = est.fit_transform(y.values.reshape(-1, 1)).astype(int).flatten()
```

```
C:\Users\HP\anaconda3\Lib\site-
packages\sklearn\preprocessing\_discretization.py:239: FutureWarning: In version
1.5 onwards, subsample=200_000 will be used by default. Set subsample explicitly
to silence this warning in the mean time. Set subsample=None to disable
subsampling explicitly.
  warnings.warn(
```

[41]:

```python
# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y_bins, test_size=0.2,␣
 ↪random_state=42)
```

[42]:

```python
# Initialize the logistic regression model
model = LogisticRegression(max_iter=1000)
```

[44]:

```python
# Train the model on the training data
model.fit(X_train, y_train)
```

[44]:

```
LogisticRegression(max_iter=1000)
```

[51]:

```python
# Predict on the testing data
y_pred = model.predict(X_test)
```

[52]:

```python
# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.5531914893617021
```

MODEL OPTIMIZATION

[53]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
```

[54]:

```
# Feature Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

[55]:

```
# Hyperparameter Tuning
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}
grid_search = GridSearchCV(LogisticRegression(max_iter=1000), param_grid, cv=5)
grid_search.fit(X_train_scaled, y_train)
```

```
C:\Users\HP\anaconda3\Lib\site-packages\sklearn\model_selection\_split.py:725:
UserWarning: The least populated class in y has only 3 members, which is less
than n_splits=5.
  warnings.warn(
C:\Users\HP\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:460:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[55]:

```
GridSearchCV(cv=5, estimator=LogisticRegression(max_iter=1000),
             param_grid={'C': [0.001, 0.01, 0.1, 1, 10, 100]})
```

[56]:

```
# Get the best model
best_model = grid_search.best_estimator_
```

[57]:

```
# Evaluate the best model
accuracy = best_model.score(X_test_scaled, y_test)
print("Accuracy:", accuracy)
print("Best hyperparameters:", grid_search.best_params_)
```

```
Accuracy: 0.9627659574468085
Best hyperparameters: {'C': 100}
```